优化实现

2020年7月6日 16:19

1. 到达定值分析
   a. 定值(Definition)：变量x的定值是(可能)**将一个值赋给x的语句**
   b. 到达定值(Reaching Definition)：如果某个变量x的一个定值d到达点p，在点p处使用的x的值可能就是由d最后赋予的
      i. 即如果存在一条从紧跟在定值d后面的点到达某一程序点p的路径，而且在此路径上d没有被“杀死”，则称定值d到达程序点p
      ii. 杀死：如果在此路径上有对变量x的其它定值d’，则称变量x被这个定值d’“杀死”了。如定值d: u= v+w“生成”了一个对变量u的定值d，并“杀死”了程序中其它对u的定值
   c. 到达定值的用途
      i. 循环不变计算的检测：如果循环中含有赋值x=y+z，而y和z所有可能的定值都在循环外面(包括y或z是常数)，那么y+z就是循环不变计算
      ii. 常量合并：如果对变量x的某次使用只有一个定值可以到达，并且该定值 把一个常量赋给x，那么可以简单地把x替换为该常量
      iii. 判定变量x在p点上是否未经定值就被引用
   d. 到达定值的传递函数
      i. f_d(x) = gen_d∪(x-kill_d)是定值语句d=v+w的传递函数
         1) gen_d={d}=语句d生成的定值的集合
         2) kill_d=由语句d杀死的定值的集合=其他u的定值
      ii. f_B(x) = gen_B∪(x-kill_B)是基本块B的传递函数
         1) kill_B=kill1∪kill2 ∪… ∪killn=被**基本块B中各个语句杀死的定值**的集合
         2) gen_B = genn ∪(genn-1 –killn) ∪(genn-2 –killn-1 –killn) ∪… ∪(gen1 –kill2 –kill3 –…–killn)=**基本块中没有被块中各语句“杀死”的定值的集合**
   e. 到达定值的数据流方程
      a. IN[B]：到达流图中基本块B的入口处的定值的集合
      b. OUT[B]：到达流图中基本块B的出口处的定值的集合
      c. 方程

         OUT[ENRTY]=Φ

         OUT[B]=fB(IN[B])(B≠ENTRY)

            联合fB(x) = genB∪(x-killB)

            得到OUT[B] = genB∪(IN[B]-killB)

         IN[B]= ∪_{P是B的一个前驱OUT[P]} (B≠ENTRY)
      d. genB和killB的值可以直接从流图计算出来，因此在方程中作为已知量
2. 到达定值方程
   a. 输入：流图G，其中每个基本块B的genB和killB都已计算出来
   b. 输出：IN[B]和OUT[B]
   c. 方法：

   OUT[ENTRY] = Φ;

   for (除ENTRY之外的每个基本块B)

      OUT[B] = Φ;

   while (某个OUT值发生了改变)

for (除ENTRY之外的每个基本块B) {
IN[B]= ∪_{P是B的一个前驱}OUT[P];
OUT[B] = genB∪(IN[B]-killB) }

d. 引用-定值链(Use-Definition Chains)UD链
   i. 是一个列表，对于变量的每一次引用，到达该引用的所有定值都在该列表中
   ii. 如果块B中变量a的引用之前有a的定值，那么只有a的最后一次定值会在该引用的ud链中
   iii. 如果块B中变量a的引用之前没有a的定值，那么a的这次引用的ud链就是IN[B]中a的定值的集合

3. 活跃变量分析
   a. 活跃变量：对于变量x和程序点p，如果在流图中沿着从p开始的**某条路径会引用变量x在p点的值**，则称变量x在点p是活跃(live)的，否则称变量x在点p不活跃(dead)
      i. 某条路径包括循环了一圈回来重新引用，所以for i++的i如果没有被重新定值，在依次循环后保持了值，就是活跃的。同理，如果循环中是常量，但会被引用，就是活跃的
   b. 活跃变量用途
      i. 删除无用赋值：如果x在点p的定值在基本块内所有后继点都不被引用，且x在基本块出口之后又是不活跃的，那么x在点p的定值就是无用的
      ii. 基本块分配寄存器：如果所有寄存器都被占用，并且还需要申请一个寄存器，则应该考虑使用已经存放了死亡值的寄存器，因为这个值不需要保存到内存。如果一个值在基本块结尾处是死的就不必在结尾处保存这个值
   c. 活跃变量的传递函数
      i. 逆向数据流问题
      ii. IN[B]= fB(OUT [B])
      iii. fB (x) = useB ∪(x-defB)
      iv. defB：在基本块B中定值，但是**定值前在B中没有被引用**的变量的集合
      v. useB：在基本块B中引用，但是**引用前在B中没有被定值**的变量集合
         1) 形如i=i+1的i出现在use中
   d. 活跃变量数据流方程
      a. IN[B]：在基本块B的入口处的活跃变量集合
      b. OUT[B]：在基本块B的出口处的活跃变量集合
      c. 方程：
         a. IN[EXIT] = Φ
         b. IN[B] = fB(OUT[B])（B≠EXIT)
            a. 结合fB(x) = useB∪(x-defB)
            b. 得到IN[B] = useB∪(OUT[B]-defB)
         c. OUT[B]= ∪_{S是B的一个后继}IN[S] （B≠EXIT)
      d. useB和defB的值可以直接从流图计算出来，因此在方程中作为已知量
   e. 活跃变量迭代算法

输入：流图G，其中每个基本块B的useB和defB都已计算出来
输出：IN[B]和OUT[B]
方法：
IN[EXIT] = Φ;
for(除EXIT之外的每个基本块B)
IN[B] = Φ;
while(某个IN值发生了改变)

```
for(除EXIT之外的每个基本块B) {
    OUT[B] = ∪_{S是B的一个后继}IN[S];
    IN[B] = useB∪(OUT[B] -defB); }
```

f. 定值-引用链(Definition-Use Chains)DU链：设变量x有一个定值d，该定值所有能够到达的引用u的集合称为x在d处的定值-引用链，简称du链
   i. 如果在求解活跃变量数据流方程中的OUT[B]时，将OUT[B]表示成从B的末尾处能够到达的引用的集合，那么，可以直接利用这些信息计算基本块B中每个变量x在其定值处的du链
   ii. 如果B中x的定值d之后有x的第一个定值d'，则d和d'之间x的所有引用构成d的du链
   iii. 如果B中x的定值d之后没有x的新的定值，则B中d之后x的所有引用以及OUT[B]中x的所有引用构成d的du链

4. 可用表达式分析
   a. 可用表达式：如果从流图的首节点到达程序点p的每条路径都对表达式x op y进行计算，并且从最后一个这样的计算到点p之间没有再次对x或y定值，那么表达式x op y在点p是可用的(available)
      i. 在点p上，x op y已经在之前被计算过，**不需要重新计算**
   b. 用途
      i. 消除全局公共子表达式
      ii. 进行复制传播
         1) 在x的引用点u用y代替x的条件：复制语句x = y在引用点u处可用
         2) 从流图的首节点到达u的每条路径都存在复制语句x = y，并且从最后一条复制语句x = y到点u之间没有再次对x或y定值
   c. 可用表达式的传递函数
      i. 对于可用表达式数据流模式而言，如果基本块B对x或者y进行了(或可能进行)定值，且以后没有重新计算x op y，则称B杀死表达式x op y。如果基本块B对x op y进行计算，并且之后没有重新定值x或y，则称B生成表达式x op y
      ii. fB(x)= e_genB∪(x-e_killB)
      iii. e_genB：基本块B所**生成的可用表达式**的集合
         1) 初始化：e_genB= Φ
         2) 顺序扫描基本块的每个语句：z= x op y
            a) 把x op y加入e_genB
            b) 从e_genB中删除和z相关的表达式
            c) 注意z可能是x或y，要先加后删
      iv. e_killB：基本块B所**杀死的U中的可用表达式**的集合
         1) 初始化：e_killB=Φ
         2) 顺序扫描基本块的每个语句：z= x op y
            a) 从e_killB中删除表达式x op y
            b) 把所有和z相关的表达式加入到e_killB中
      v. U：所有出现在程序中一个或多个语句的右部的表达式的全集
   d. 可用表达式的数据流方程
      a. IN[B]：在B的入口处可用的U中的表达式集合

b. OUT[B]：在B的出口处可用的U中的表达式集合
c. 方程
    a. OUT[ENTRY]= Φ
    b. OUT[B]=fB (IN[B]) （B≠ENTRY)
        a. fB(x)= e_genB∪(x-e_killB)
    c. IN[B]= ∩_{P是B的一个前驱}OUT[P]（B≠ENTRY)
d. e_genB 和e_killB 的值可以直接从流图计算出来，因此在方程中作为已知量

e. 计算可用表达式的迭代算法
a. 输入：流图G，其中每个基本块B的e_genB和e_killB都已计算出来
b. 输出：IN[B]和OUT[B]
c. 方法：

```
OUT[ENTRY] = Φ;
for(除ENTRY之外的每个基本块B)
        OUT[B] = U;
while(某个OUT值发生了改变)
        for (除ENTRY之外的每个基本块B) {
                IN[B]=∩_{P是B的一个前驱}OUT[P]
                OUT[B]= e_genB∪(IN[B]-e_killB); }
```

5. 支配结点和回边
a. 支配结点(Dominators)：从流图的入口结点到结点n的每条路径都经过结点d，则称结点d支配(dominate)结点n，记为d dom n
    i. 每个结点都支配它自己
    ii. 支配结点树(DominatorTree)上每个结点只支配它和它的后代结点
    iii. 直接支配结点(Immediate Dominator)：从入口结点到达n的所有路径上，结点n的最后一个支配结点称为直接支配结点。可能是支配树上的父节点
b. 寻找支配结点的数据流方程
a. IN[B]：在基本块B入口处的支配结点集合
b. OUT[B]：在基本块B出口处的支配结点集合
c. 方程

```
OUT[ENTRY] ={ ENTRY}
OUT[B] = IN[B]∪{B} （B≠ENTRY)
IN[B] = ∩_{P是B的一个前驱} OUT[P] (B≠ENTRY)
```

c. 计算支配结点的迭代算法
a. 输入：流图G，G的结点集是N，边集是E，入口结点是ENTRY
b. 输出：对于N中的各个结点n，给出D(n)，即支配n的所有结点的集合
c. 方法:

```
OUT[ENTRY]={ENTRY}
for(除ENTRY之外的每个基本块B)
        OUT[B]=N
while(某个OUT值发生了改变)
        for(除ENTRY之外的每个基本块B) {
                IN[B]=∩_{P是B的一个前驱} OUT[P]
                OUT[B]=IN[B]∪{B} }
```

d. 回边(Back Edges)：假定流图中存在两个结点d和n满足d dom n。如果存在从结点n到d的有向边n→d，那么这条边称为回边

6. 自然循环及其识别
a. 从程序分析的角度来看，循环在代码中以什么形式出现 并不重要，重要的是它是否具有易于优化的性质。自然循环就适合优化
b. 自然循环(Natural Loops)：是满足以下性质的循环
    i. 有唯一的入口结点，称为首结点(header)。首结点支配循环中的所有结点，

否则，它就不会成为循环的唯一入口

ii. 循环中至少有一条返回首结点的路径，否则，控制就不可能从“循环”中直接回到循环头，也就无法构成循环

c. 识别自然循环：给定一个回边n → d，该回边的自然循环为：d，以及所有可以不经过d而到达n的结点。d为该循环的首结点

d. 性质：除非两个自然循环的首结点相同，否则，它们或者互不相交，或者一个完全包含(嵌入)在另外一个里面

e. 最内循环(Innermost Loops): 不包含其它循环的循环。如果两个相同首结点的循环，可以合并成一个，视作最内循环

7. 删除全局公共子表达式

a. 可用表达式的数据流问题可以帮助确定位于流图中p点的表达式是否为全局公共子表达式

i. 输入：带有可用表达式信息的流图

ii. 输出：修正后的流图

iii. 方法：

对于语句s：z=x op y，如果x op y在s之前可用，那么执行如下步骤：

a. 从 s开始逆向搜索，但不穿过任何计算了x op y的块，找到所有离 s最近的计算了x op y的语句

b. 建立新的临时变量u

c. 把步骤a.中找到的语句w = x op y用下列语句代替：

u= x op y； w= u

d. 用z = u替代s

b. 删除复制语句

i. 对于复制语句s:x=y，如果在x的所有引用点都可以用对y的引用代替对x的引用(复制传播)，那么可以删除复制语句x=y

c. 在x的引用点u用y代替x(复制传播)的条件：复制语句s:x=y在u点“可用”

i. 输入：流图G、du链、各基本块B入口处的可用复制语句集合

ii. 输出：修改后的流图

iii. 方法：

对于每个复制语句x=y，执行下列步骤

a. 根据du链找出该定值所能够到达的那些对x的引用

b. 确定是否对于每个这样的引用，x=y都在IN[B]中(B是包 含这个引用的基本块)，并且B中该引用的前面没有x或者y 的定值

c. 如果x=y满足第b.步的条件，删除x=y，且把步骤a.中找 到的对x的引用用y代替

8. 代码移动

a. 循环不变计算检测算法

i. 输入：循环L，每个三地址指令的ud链

ii. 输出：L的循环不变计算语句

iii. 方法

a. 将下面这样的语句标记为“不变”：语句的运算分量或者是常数，或者其所有定值点都在循环L外部

b. 重复执行步骤c.，直到某次没有新的语句可标记为“不变”为止

c. 将下面这样的语句标记为“不变”：先前没有被标记过，且所有运算分量或者是常数，或者其所有定值点都在循环L外部，或者只有一个到达定值，该定值是循环中已经被标记为“不变”的语句

b. 代码外提

i. 前置首结点(preheader)：循环不变计算将被移至首结点之前，为此创建一个称为前置首结点的新块

ii. 前置首结点的唯一后继是首结点，并且原来从循环L外到达L首结点的边都改

成进入前置首结点。从循环L里面到达首结点的边不变

c. 循环不变计算语句s :x = y + z 移动的条件
   i. s所在的基本块是循环所有出口结点(有后继结点在循环外的结点)的支配结点
   ii. 循环中没有其它语句对x赋值
   iii. 循环中对x的引用仅由s到达
d. 代码移动算法
   i. 输入：循环L、ud链、支配结点信息
   ii. 输出：修改后的循环
   iii. 方法：
      a. 寻找循环不变计算
      b. 对于步骤a.中找到的每个循环不变计算，检查是否满足上面的三个 条件
      c. 按照循环不变计算找出的次序，把所找到的满足上述条件的循环不变计算外提到前置首结点中。如果循环不变计算有分量在循环中定值，只有将定值点外提后，该循环不变计算才可以外提

9. 作用于归纳变量的强度削弱
   a. 归纳变量：果存在一个正的或负的常量c ，使得每次x被赋值时，它的值总是增加c，则称x为归纳变量
      i. 如果循环L中的变量i只有形如i=i+c的定值(c是常量)，则称i为循环L的基本归纳变量
      ii. 每个归纳变量都关联一个三元组。如果j = c×i+d，其中i是基 本归纳变量，c和d是常量，则与j相关联的三元组是(i, c, d)
   b. 归纳变量检测
      i. 输入：带有循环不变计算信息和到达定值信息的循环L
      ii. 输出：一组归纳变量
      iii. 方法：
      iv. 扫描L的语句，找出所有基本归纳变量。在此要用到循环不变计算信息。与每个基本归纳变量i相关联的三元组是(i,1, 0)
      v. 寻找L中只有一次定值的变量k，它具有下面的形式： k=c′×j+d′。其中c′和d′是常量，j是基本的或非基本的归纳变量
         1) 如果j是基本归纳变量，那么k属于j族。k对应的三元组可以通过其定值语句确定
         2) 如果j不是基本归纳变量，假设其属于i族，k的三元组可以通过j的三元组和k的定值语句来计算，此时我们还要求：
            a. 循环L中对j的唯一定值和对k的定值之间没有对i的定值
            b. 循环L外没有j的定值可以到达k

10. 归纳变量的删除